OOPs

What is OOP’s?

Object-Oriented Programming language ,it is not a separate programming language but it is a part of programming. OOPs refers to a language that uses objects in programming as clear by its name .
The main concept of oops is used to bind the data and functions that work together in a single unit .
OOPs are based on real world entities like inheritance ,polymorphism etc.

OOPs Main concepts

1. Classes
2. Objects
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Advantages of OOPs

- Reusability : OOPs encourages the reuse of existing code
- Scalability : OOPs makes it easy to scale a program to handle big amount of work to split down into breaks (In sub work)
- Maintainability: OOPs makes it easy to maintain the code and also to fix bugs
- Portability: OOPs allow our code can easily ported on different platforms

Disadvantages of OOPs

- Everything is treated as object in OOPs so before applying it we need to have excellent thinking in terms of objects
- We can not apply OOPs everywhere as it is not a universal language . it is not suitable for all types of problems
- The length of program using OOPs is much larger than the procedural approach

What are Classes in OOPS?

A class is a user defined data type . A class contains properties like variables and methods and to use it ,we have to create an object.
It represents a set of properties or methods which are common to all objects of one type.
It is like a blueprint for an object
Syntax: To define a class we use class keyword

```
 Class classname:
          Variables
          Methods
          Comments
```

Example : consider a class of car.They are many cars with different names and brands but all of them will share some common properties like all of them will have four wheels ,Speed Limit,Mileage range etc . So here, a car is a class and wheels,Speed limit ,mileage are their properties.

What are objects in OOPs?

An object is an instance of class . It is a basic unit of OOPs and represent the real life entities .When class is defined ,no memory is allocated but when it is instantiated(eg. an Object is created) memory is allocated .An object has an identity ,state and behavior
After creating a class to use the class we have to create an object
Example :

```
Class myname:
           x=5                                          //variable

p1=myname()                                         //p1 is the object of myname class
print(p1.x)                                         //output =5
```

What are constructors and where do we use it ?

Constructors are used in classes to declare the instance variables in class.
constructor have fixed in python __init__
It is automatically called when object is created (executes only once per object)

Constructors are basically a function but it is a special function which is only used in classes.

Example:

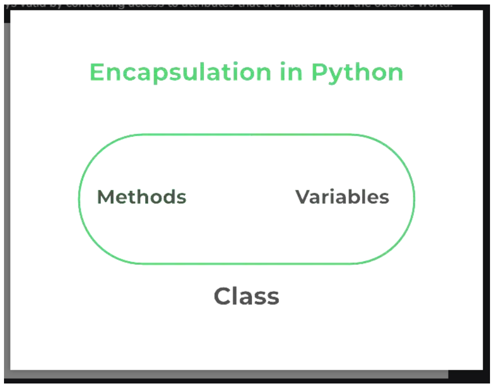
```
Class car:                                      //class (car)
    Def __init__(self):                         //constructor
            self .name ="yash"                  //instance variables
            self .rollno=19239
    Def show(self):                             //instance methods
            print("your name is :" , self.name)
            print("your rollno is :" ,self.rollno)
objref=car()
objref.show()
```

```
Output = your name is yash
         your rollno is 19239
```

Note: self keyword is used to represent an instance (object) of given class

What is Encapsulation in OOPS?

Encapsulation is one of the fundamental concepts of OOPS . Using classes and objects we do encapsulation . It describes the idea of wrapping data and the methods that work on data within one unit.Using encapsulation,we can hide an object's internal representation from the outside .This is called information hiding.



Example : consider an example , In a company ,there are different sections like the account section ,finance section ,sales section etc. The financial section handle all the financial records .similarly sales section handle all sales records.Now their is a situation Where the sales section needs a specific month's financial record .In this case sales section is not allowed to directly access the data of the financial section.Sales section will first have to contact some other officer in the financial section and then request him to give the particular data .This is what is Encapsulation.

This is simple program using encapsulation

```
class BankAccount:
    def __init__(self):
        self.__balance = 0

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount > self.__balance:
            print("Insufficient balance")
        else:
            self.__balance -= amount

    def get_balance(self):
        return self.__balance

b = BankAccount()
b.deposit(1000)
b.withdraw(500)
print(b.get_balance()) # this will print 500
```

Access modifier

Encapsulation can be achieved by declaring the data member and methods of a class as private and protected .But in python , we don't have direct access modifiers like public,private and protected. We can achieved this by using single underscore and double underscore

Example of private member can not accessible :

```
Class emp:
 class Employee:
    # constructor
    def __init__(self, name, salary):
        # public data member
        self.name = name
        # private member
        self.__salary = salary
```

```
# creating object of a class
emp = Employee('Jessa', 10000)

# accessing private data members
print('Salary:', emp.__salary)

output= attribute error
```

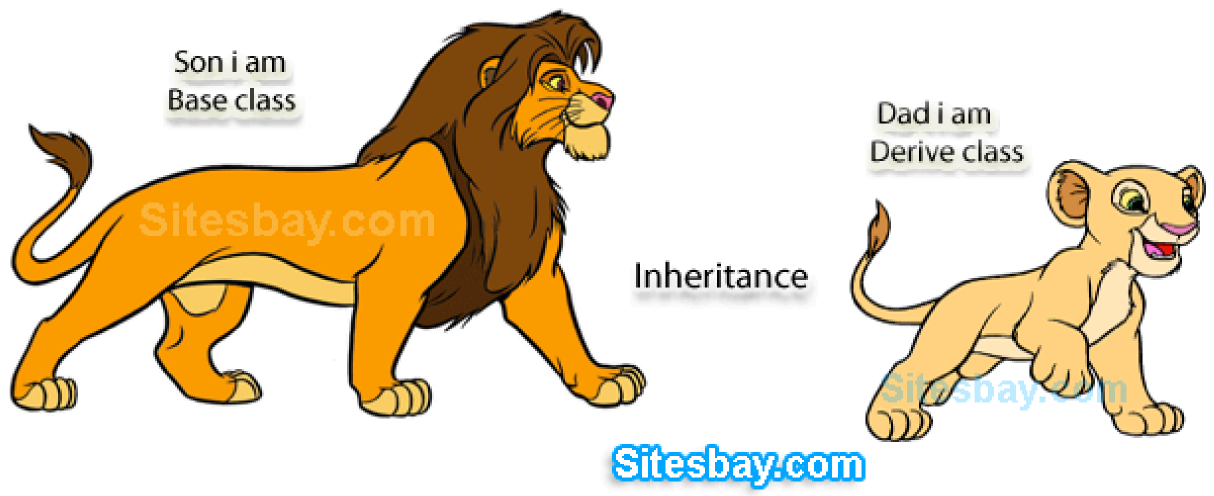
Example to access private member using instance method

```
class Employee:
    # constructor
    def __init__(self, name, salary):
        # public data member
        self.name = name
        # private member
        self.__salary = salary

    # public instance methods
    def show(self):
        # private members are accessible from a class
        print("Name: ", self.name, 'Salary:', self.__salary)

# creating object of a class
emp = Employee('Jessa', 10000)

# calling public method of the class
emp.show()
```

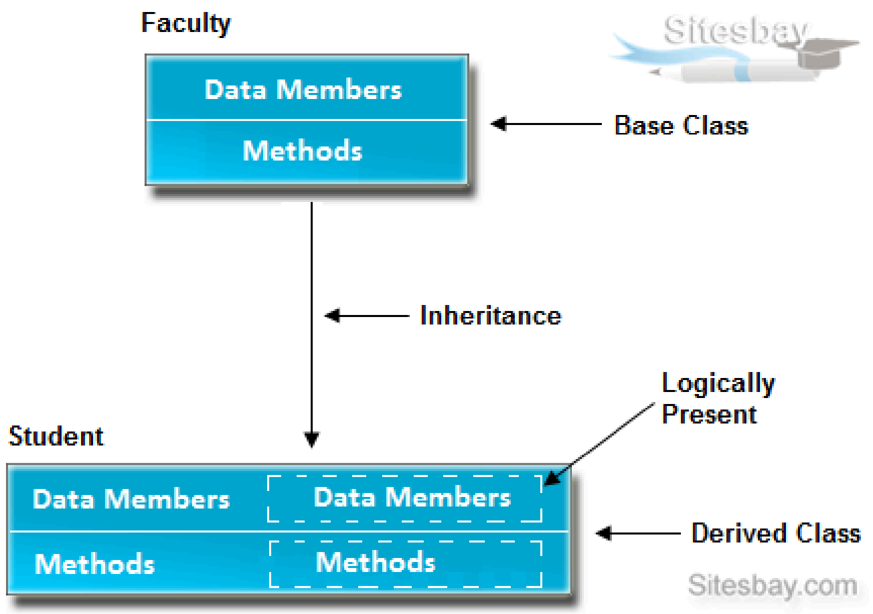
What is inheritance in OOPs?

Getting data and properties from parent class to child class is called inheritance .
As it name clear that inherit from one thing to another thing
In inheritance we don't need to create parent class object to use parent class object we can create child class object

Example of inheritance:



The following diagram use view about inheritance.



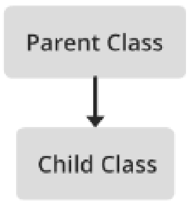
Syntax of inheritance:

```
Class baseclass:
    {body}
Class childclass(baseclass):
     {body}
```

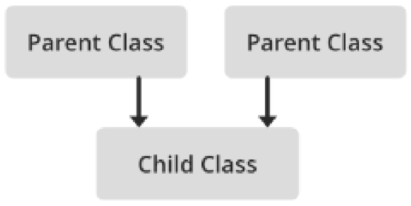
Simple Program to demonstrate inheritance

```
Class Pclass:
     Def __init__(self):
           self.name=”yash”
           self.rollnumber=1302294
     Def show(self):
           print(“your name is :”,self.name)
           print(“your rollnumber is :”,self.rollnumber)
Class cclass(Pclass):
      Def display(self):
          print(“this is your child class “)

objref=cclass()
objref.show()              //this is parent class method
objref.display()           //this is child class method
```

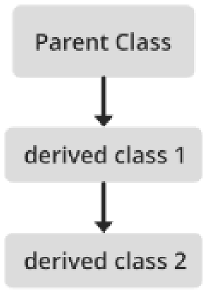
Types of inheritance



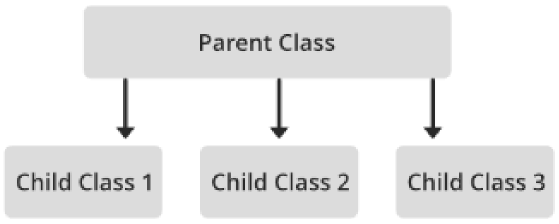
Single inheritance



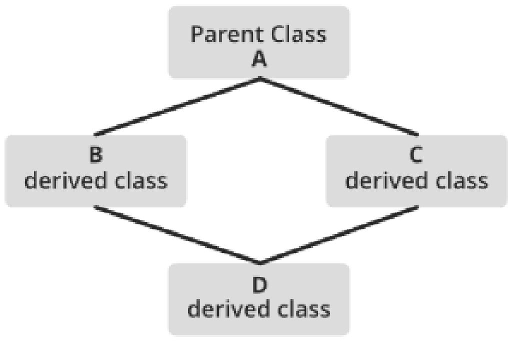
Multiple inheritance



Multilevel inheritance



Hierarchical inheritance



Hybrid Class

Super function in Inheritance?

Super function is used to refer to the parent class or superclass.it allows you to call methods defined in superclass from the subclass .When child class constructor is available it does not execute base class constructor so to do that we use super function .

To use super functions we have a super keyword.

Example of inheritance without super function

```
class Person:

    # Constructor
    def __init__(self, name, id):
        self.name = name
        self.id = id

    # To check if this person is an employee
    def Display(self):
        print(self.name, self.id)


class Emp(Person):

    def __init__(self, name, id):

        self.name_ = name


    def Print(self):
        print("Emp class called")

Emp_details = Emp("Mayank", 103)

# calling parent class function
print(Emp_details.name, Emp_details.name_)
```

```
Output = attribute error
```

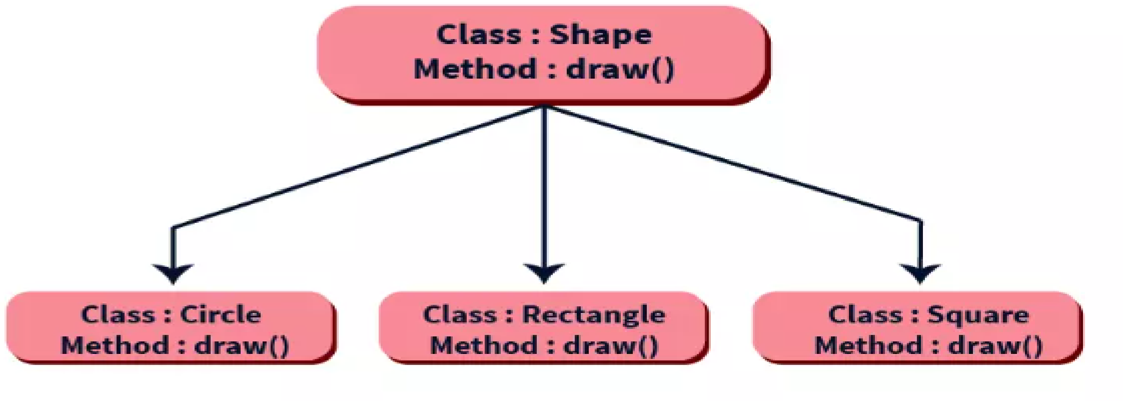
TO solve this problem we use super function

```
class Person:
    # Constructor
    def __init__(self, name, id):
        self.name = name
        self.id = id

    # To check if this person is an employee
    def Display(self):
        print(self.name, self.id)


class Emp(Person):

    def __init__(self, name, id):
        self.name_ = name
        super().__init__(name, id)

    def Print(self):
        print("Emp class called")

Emp_details = Emp("Yash", 103)

# calling parent class function
print(Emp_details.name_, Emp_details.name)


Output = Yash Yash
```

What is polymorphism in OOPS ?

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form
In Python, polymorphisms refer to the occurrence of something in multiple forms.

The ability of one function to perform multiple functionalities. It can be implemented in different ways, including operator overloading(example that + this can be concatenation in string and also addition in numbers), built-in functions, classes, and inheritance. In operator overloading, the same operator is used in different ways, such as addition and string concatenation polymorphism is implemented in Python for a variety of purposes, most Operator and Method overloading, and Method overriding. This polymorphism may be accomplished in two distinct ways: overloading and overriding.



Real life Example of polymorphism :
A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person possesses different behavior in different situations. This is called polymorphism.

Method overloading?

Method Overriding is another crucial concept in OOP) that empowers subclasses to provide a specific implementation for a method already defined in their superclass.(redefining the methods in base class which are defined in superclass)

When a method is overridden, the version defined in the subclass takes precedence over the one in the superclass. This principle of polymorphism enables objects of the subclass to be used interchangeably with objects of the superclass. This assists in promoting code extensibility and flexibility.

Method Overriding is particularly useful in scenarios where a subclass needs to enhance or modify the functionality inherited from its superclass while maintaining the overall structure and interface. By doing so, developers can efficiently reuse existing code and create specialized classes that build upon the foundation of more general ones.

Consider the following code example:

```
class Vehicle:
    def run(self):
        print("Saves Energy")
class EV(Vehicle):
    def run(self):
    super().run()//super function is used to call the method of
base class
        print("Run on Electricity")
ev = EV()
ev.run()

output= save energy
        Run on Electricity
```

Method overloading?

Method Overloading is a fundamental concept in OOP that enables a class to define multiple methods with the same name but different parameters. (two or more methods with same name in same class but different signature like argument number or datatype of that arguments)

When a method is overloaded, Python determines which version of the method is to be executed based on the arguments provided during the function call. By using Method Overloading, programmers can create cleaner and more concise code. It is because related functionalities can be grouped under the same method name.

Moreover, Method Overloading empowers developers to build flexible and adaptive solutions.

Consider the following code example:

```
class MathOperations:
    def add(self, a, b):

        return a + b



    def add(self, a, b, c):

        return a + b + c



    def add(self, a, b, d):
        return a+b+d

obj=MathOperations()
print(obj.add(12,2,3))

output= 17
```

Note : To do method overloading we can use variable length arguments

What is abstraction in OOPs?

Abstraction is the process of hiding the internal details of an application from the outer world. Abstraction is used to describe things in simple terms. It’s used to create a boundary between the application and the client programs.

Real life example of abstraction :

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

Abstract classes

Abstract class is the class that contains at least one abstract method and we can not create an object for abstract class because it is incomplete.abstract classes inherit from abc module so first we need to import abc module

Abstract method
A method without body or implementation is called an abstract method.
To create abstract method we need to use @abstractmethod and in abstract method we need to pass it

Program to demo abstraction in OOPs:

```
# Import required modules
from abc import ABC, abstractmethod

# Create Abstract base class
class Car(ABC):
    def __init__(self, brand, model, year):
             self.brand = brand
             self.model = model
             self.year = year

# Create abstract method
    @abstractmethod
    def printDetails(self):
             pass

# Create concrete method
    def accelerate(self):
             print("speed up ...")

    def break_applied(self):
         print("Car stop")

# Create a child class
class Hatchback(Car):
   def printDetails(self):
         print("Brand:", self.brand);
         print("Model:", self.model);
         print("Year:", self.year);

   def Sunroof(self):
         print("Not having this feature")

# Create a child class
class Suv(Car):

    def printDetails(self):
         print("Brand:", self.brand);
         print("Model:", self.model);
         print("Year:", self.year);

    def Sunroof(self):
         print("Available")


car1 = Suv("Maruti", "Alto", "2022");

car1.printDetails()
car1.accelerate()
car1.Sunroof()

Output =  Brand: Maruti
          Model: Alto
          Year: 2022
          speed up ...
          Available
```